

Name : Mostafa Kermaninia

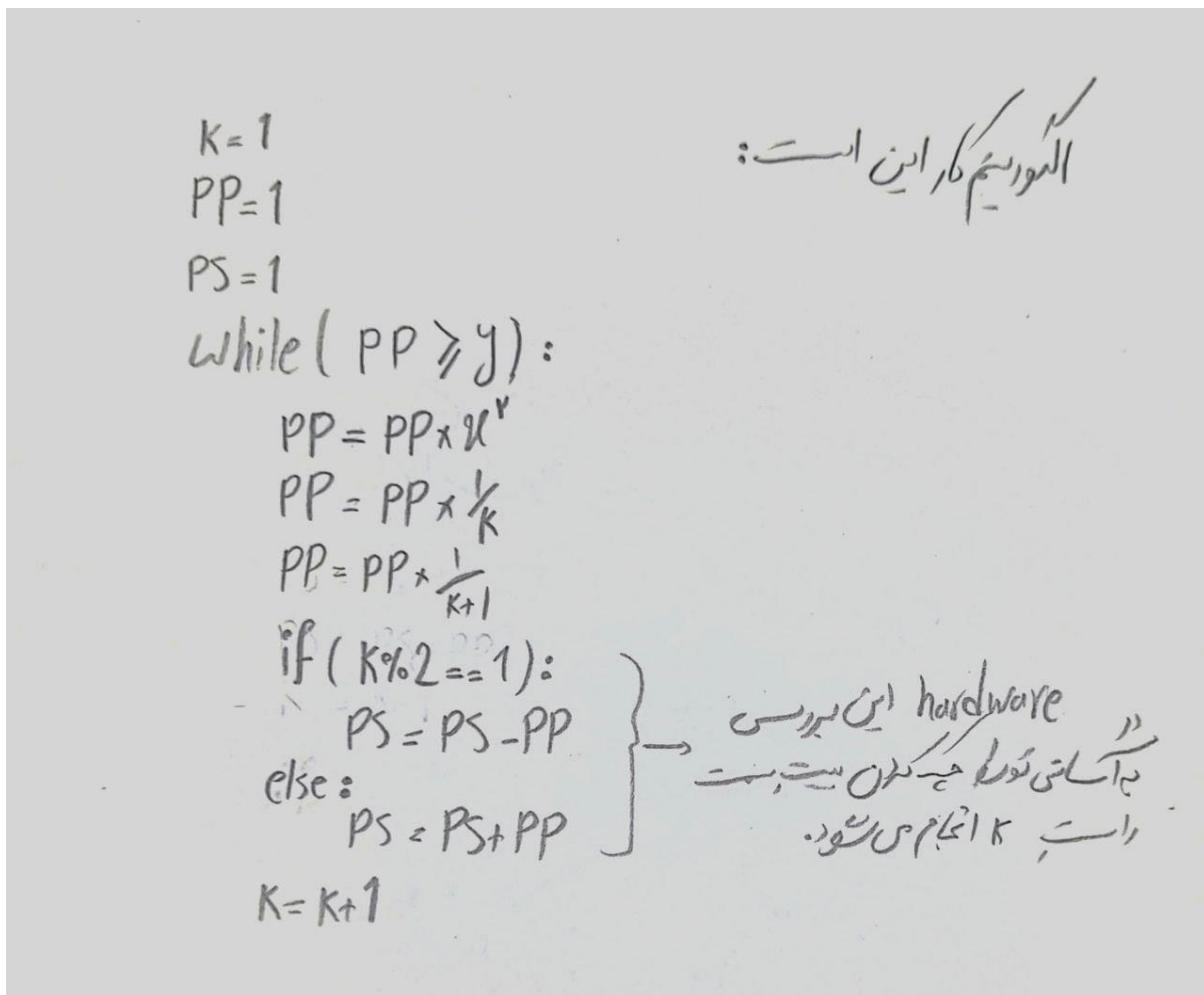
SID : 810101575

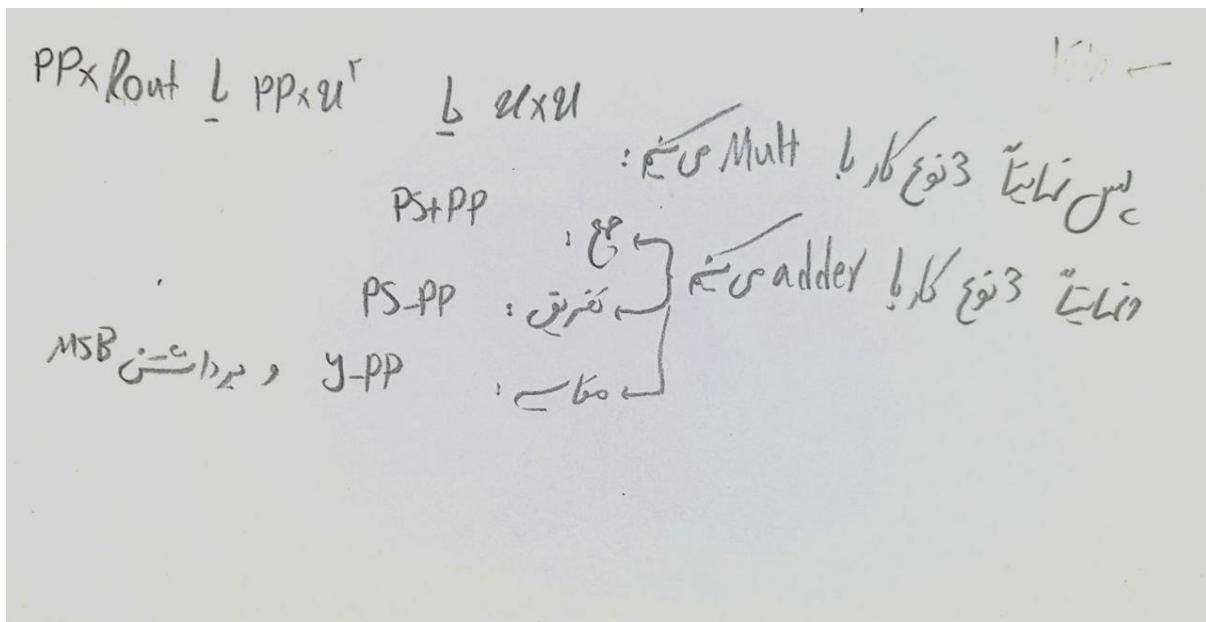
Course name : Introduction to Digital System Design

Course number : 4021810136701

ابتدا پیش از هر کاری دیدگاه اولیه‌ی خودم را پیاده سازی می‌کنم و طبق مراحلی که در تاپیک 10 گفته شد، ابتدا **controller** و سپس **datapath** را طراحی اولیه می‌کنم:

شرح ابتدایی:





DATA PATH

با توجه به شکل بسط تیلور اولا ما به یک **ADDER** نیازمندیم برای جمع زدن **term**ها. که چون نهایتا پاسخ مدنظرمان یک عدد 16 بیتی است که بخش int آن همواره صفر است با توجه به متن سوال و در نتیجه نزدیکترین عدد 1 بصورت 00000000.11111111 خواهد شد و منطقی هم هست چون نهایتا دقت ما توسط y تعیین میشود که 8 بیت اعشار دارد پس هیچگاه به عدد 1 کامل نمیرسیم و درست است که **term** های مثبت هم داریم اما با توجه به شکل **term**ها، حاصل **partial sum** هیچگاه از 1 بیشتر نمیشود حتی اگر $i = p$ باشد. پس کلا میتوان هر جا قرار بود نتیجه 16 بیتی بشود، ادر و multp را 8 بیتی گذاشته و کار کرد، اما برای جلوگیری از ابهام و خطاهای احتمالی، 16 بیتی میگذارم.

دوما یک **array multiplier** میخواهیم برای ساختن **term**ها که آن هم هر دفعه یا x^2 را در ضرب میکند و یا $1/k$ را ضرب میکند. پس ورودی هایش را نمیتوان دو تا 8 بیتی گذاشت، پس $k/1$ ها را هم از سمت چپ با افزودن 8 بیت صفر تبدیل به 16 بیت کرده و بهمراه که آن هم 16 بیت است ضرب میکنیم. و حاصل ضرب را هم هر بار 16 بیت وسطش که به نزدیک است جدا کرده و در محاسبات استفاده میکنیم.

ایده: بجای array multiplier میتوان گذاشت اما بنظرم اولاً یوینت

سوال این نیست که آنرا هم درون این سیستم طراحی کنیم، دوماً تعداد زیادی clk از دست

میدهیم در برابر اینکه hardware سبکتری استفاده میشود ولی اینجا بنظرم نمیصرفد

از طرف دیگر چون هم جمع زدن و هم ضرب کردن بطور sequential رخ میدهد کلا فقط به یک ادر و یک مالتیپلایر نیازمندیم.

از طرفی طبق گفته‌ی صورت سوال برای محاسبه‌ی $1/k$ میتوان از یک ROM استفاده کرد که تعداد خانه‌های آنرا با محاسباتی که انجام خواهیم داد مساوی 16 میکنیم، از طرفی چون قرار است بعداً آنها را در $2^8 \times 2^8$ ضرب کنیم که 15 بیتی است، یا باید از همینجا 16 بیتی کنیم و یا اونجا 8 بیت صفر به سمت راستش اضافه کنیم، پس همان 8 بیتی میگیریم.

از طرفی یکسری رجیستر برای ذخیره‌ی X, Y و یکی برای partial multiply term و یکی برای partial sum term

ایده: میتوان بجای خود X، همان ابتدا X به توان دو را ذخیره کرد چون در partial multiply

ایکس‌ها دوتا دوتا ضرب میشوند، همانکار را هم برای ROM میتوان کرد یعنی چون هر term

دفعه در term‌های جدید، $1/i * 1/(i+1)$ ضرب میشود پس میتوان یک Rom با این عبارت ها

ساخت.

یک TFF هم بنظرم لازم است برای تشخیص اینکه موقع جمع زدن، باید term را جمع کنیم یا تفریق کنیم (برای تفریق نیاز به subtractor هم نیست چون با همان ادر میتوان تفریق را پیاده کرد)

ایده: میتوان برای فهمیدن مثبت یا منفی بودن term ای که جمع میکنیم، بیت سمت راست همان ای که برای یافت خانه‌ی مدنظر در ROM بکار میرود استفاده کنیم.

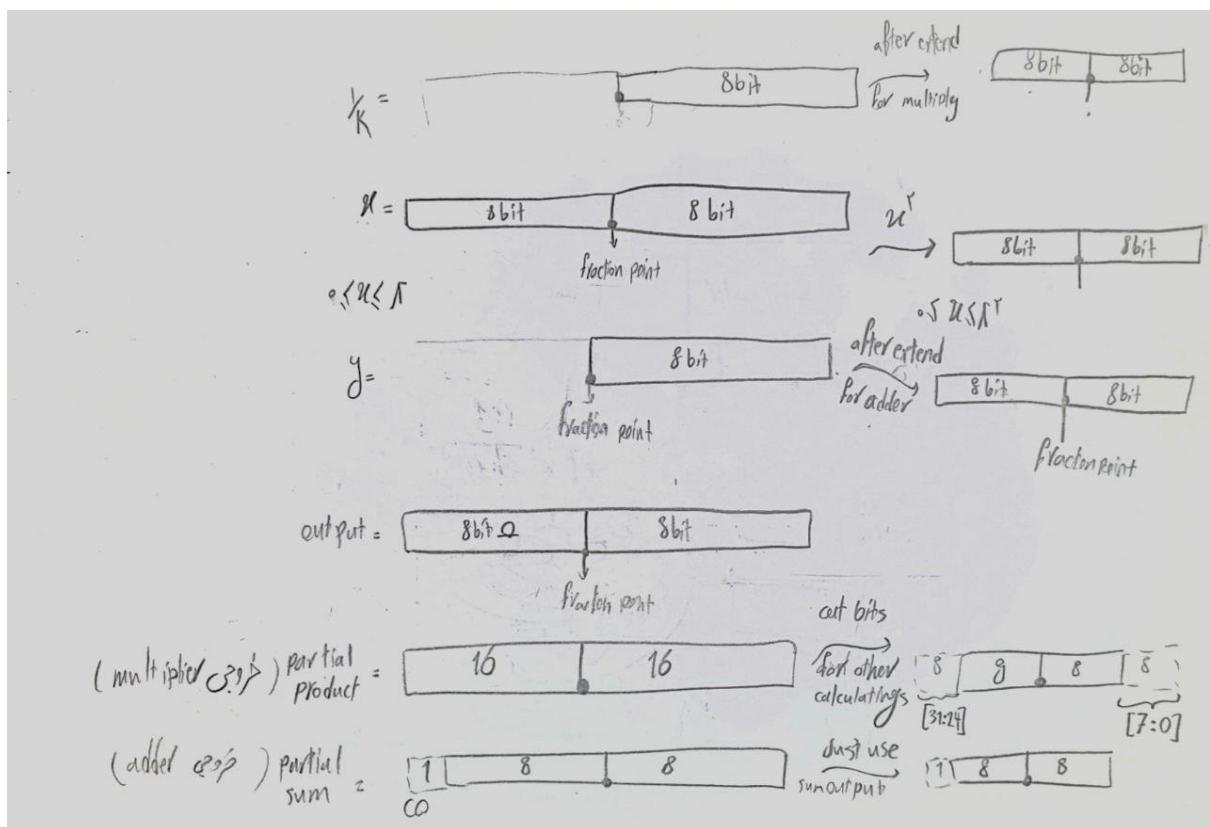
نهایتاً هم یک counter میخواهیم که از شروع محاسبه آنرا count up کنیم تا ادرسی که قرار است از ROM بخوانیم را به ما بگوید.

سپس برای چک کردن اینکه در هر مرحله دقتمن از a کمتر شده یا نه، از یک comparator استفاده میکنیم.

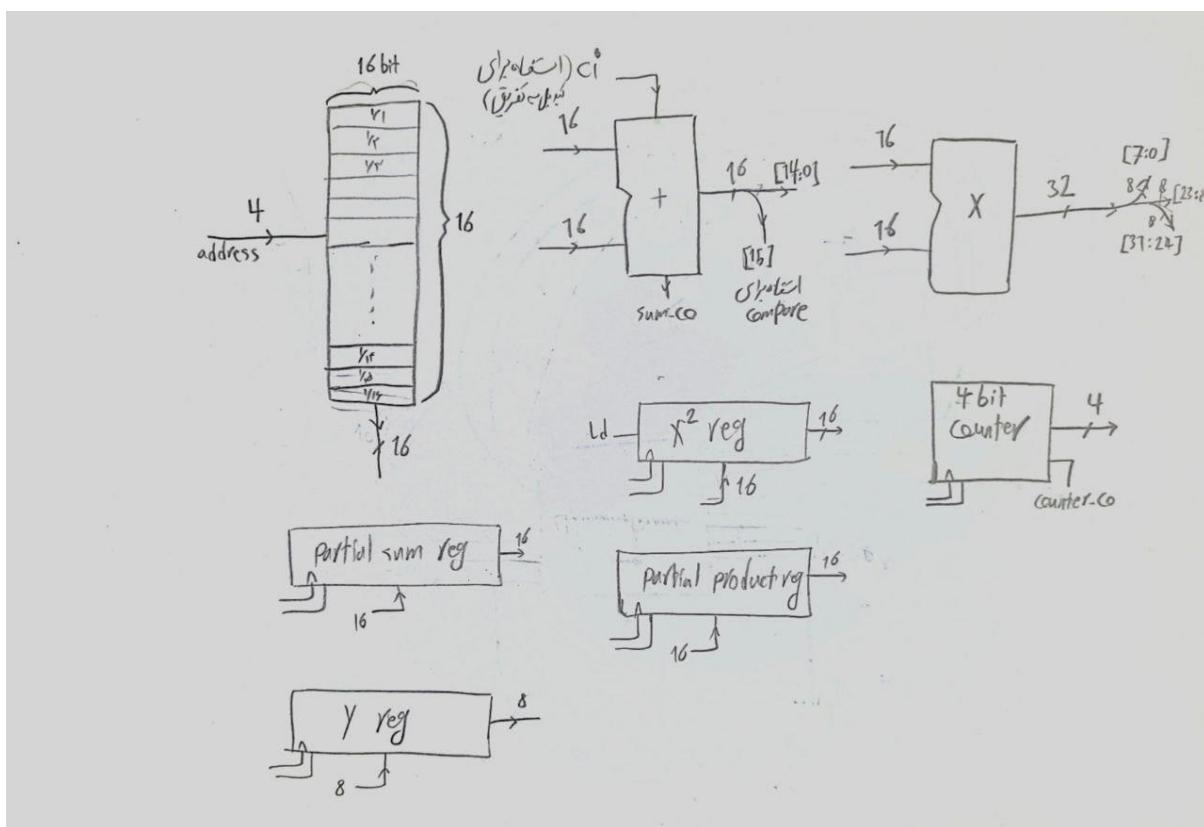
ایده: میتوان بجای **comparator** از همان ادرای استفاده کرد اینجوری که هر بار از مقدار خود term بدون اعمال مثبت و منفی، u را کم کنیم و اگر حاصل نامنفی بود، ادامه دهیم به عملیات محاسبه.

نهایتا همچین چیزی داریم:

روند extend کردن بیت ها

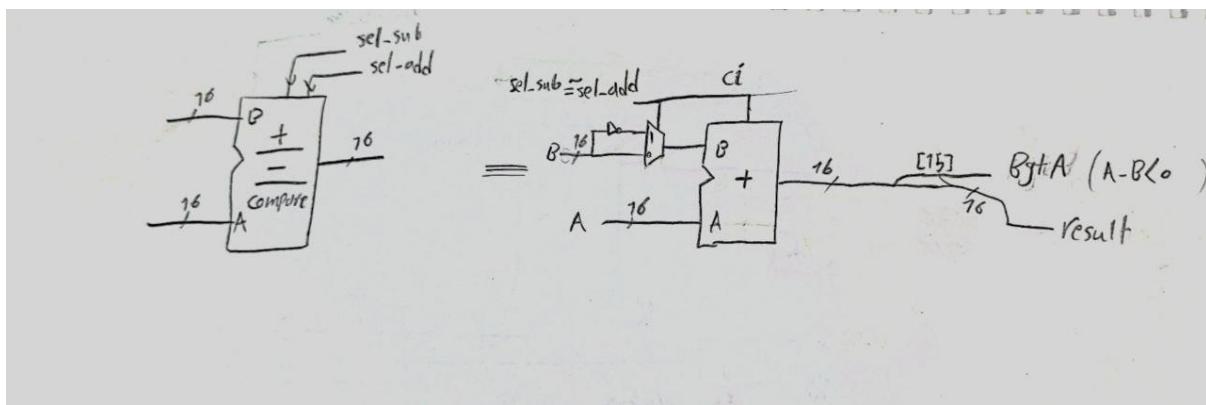
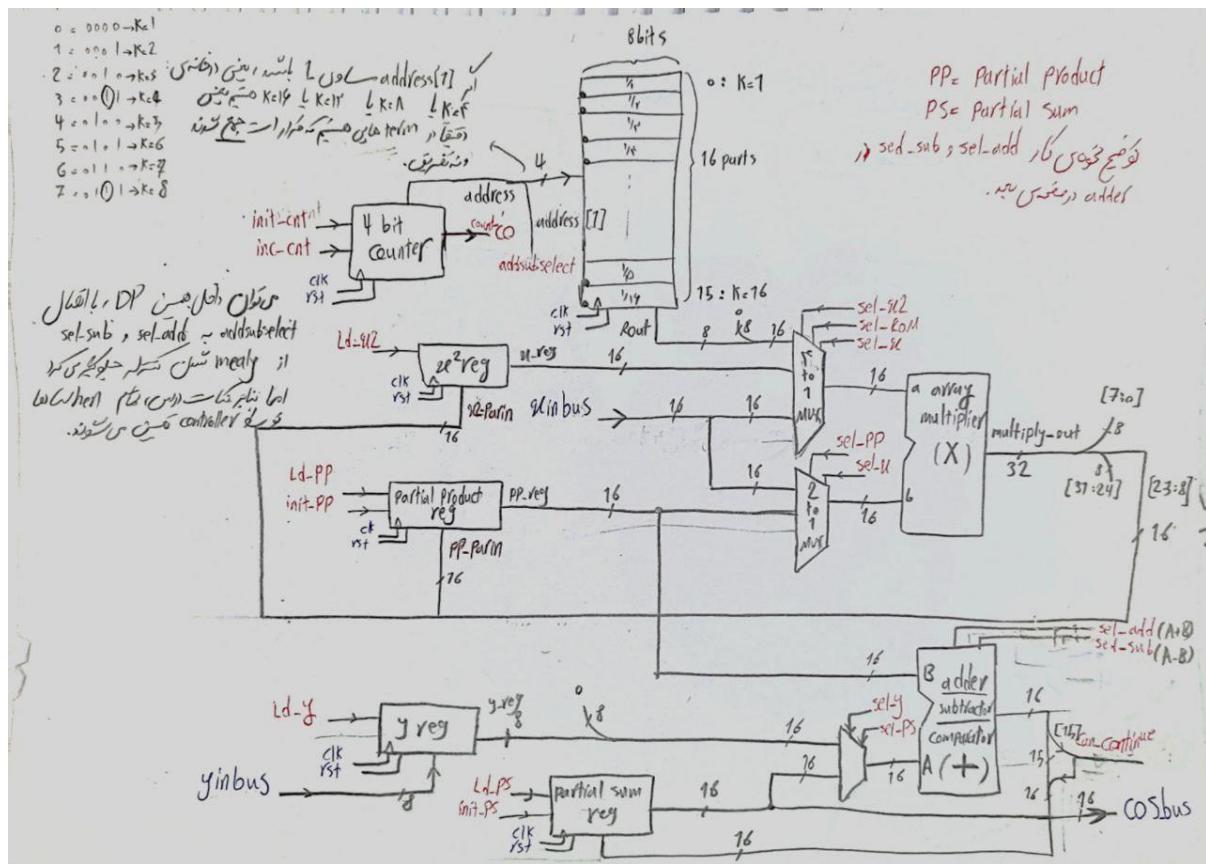


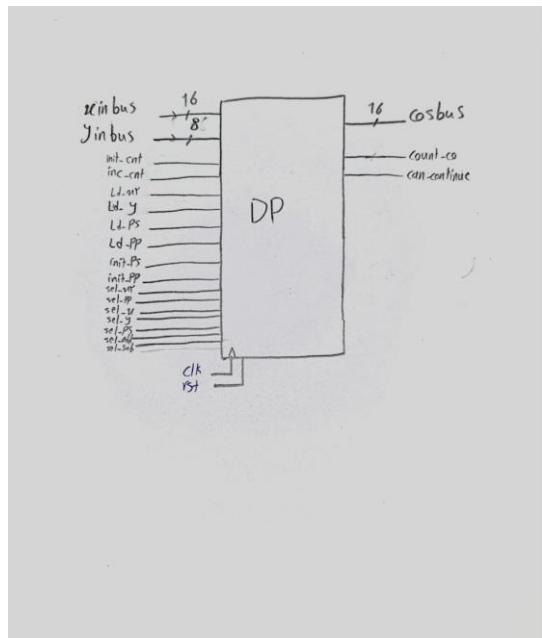
کامپوننت های bussing بدون DP



BUSSING

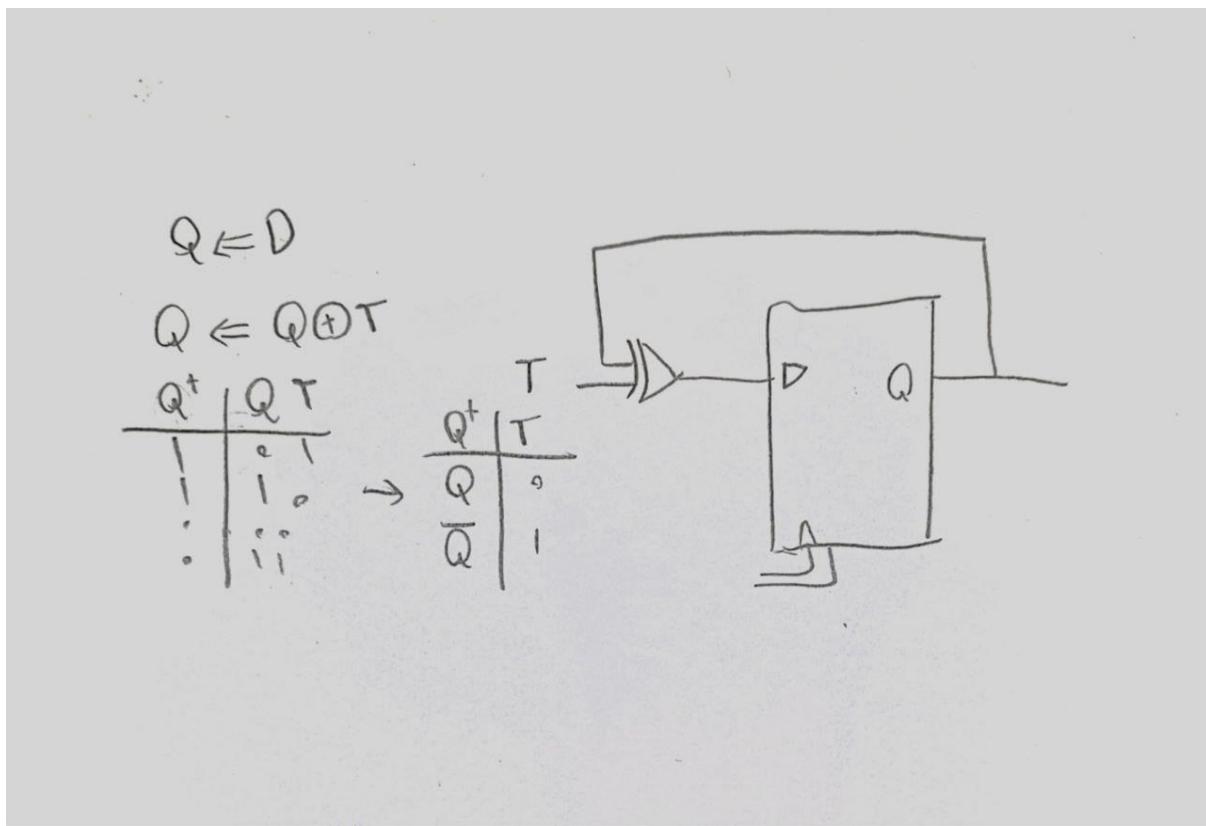
در این بخش یسری سیم و MUX و این ها اضافه کرده و DP نهایی این شد:





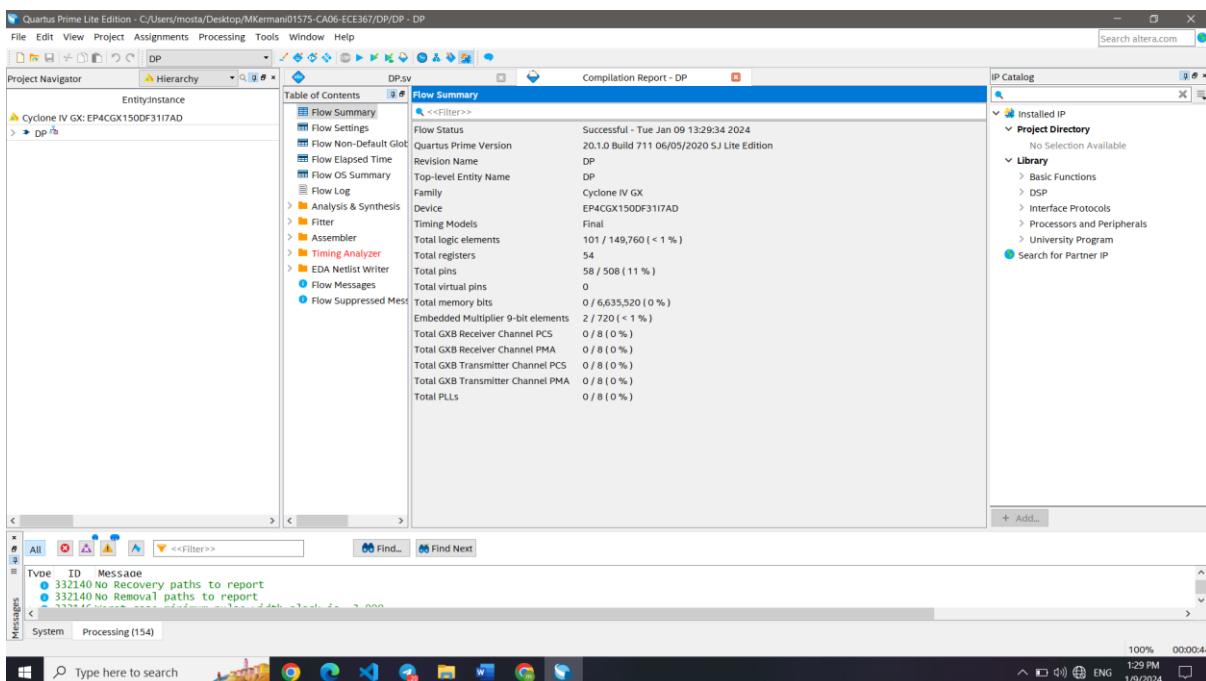
DATAPATH

دانه دانه کامپوننت ها را وریلاگشان را میزندیم و تست میکنیم تا نهایتا این ها بدست میابند(ساختار را مقداری ساده کردم یعنی مثلا برای مقایسه مستقیما **comparator** گذاشتم و برای مثبت و منفی کردن در جمع بجای استقاده از **mealy** در کنترلر، مستقیما با یک **toggle** کار را جمع کردم(در ابتدا آنرا صفر کرده و هر بار **toggle** میکند و ساختار آن هم ساده است و با یک **DFF** براحتی هندل میشود)

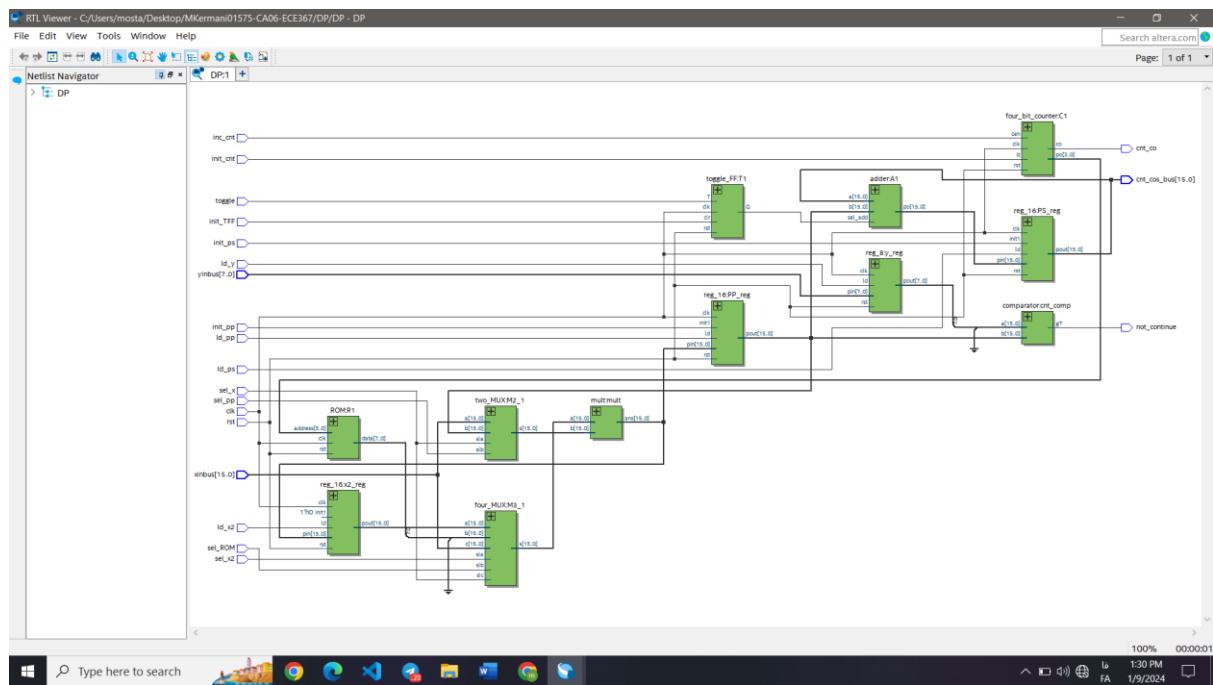


و بعد از سیمولیت در کوارتز داریم:

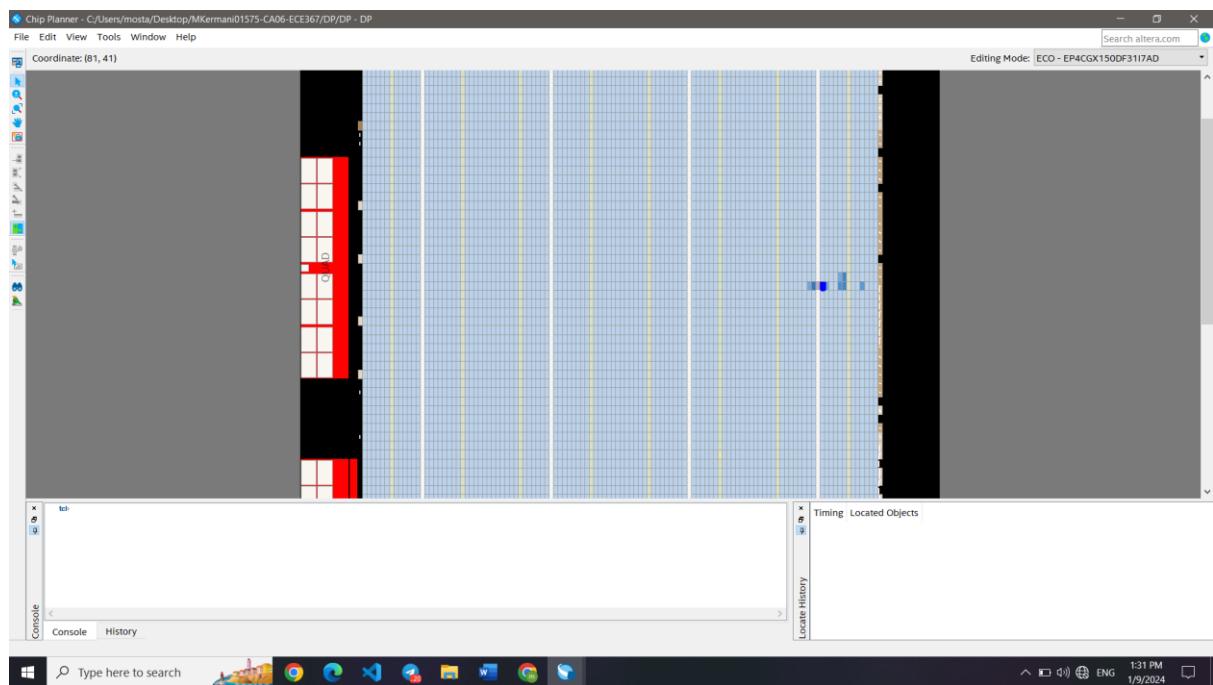
اطلاعات کلی:



RTL viewer:



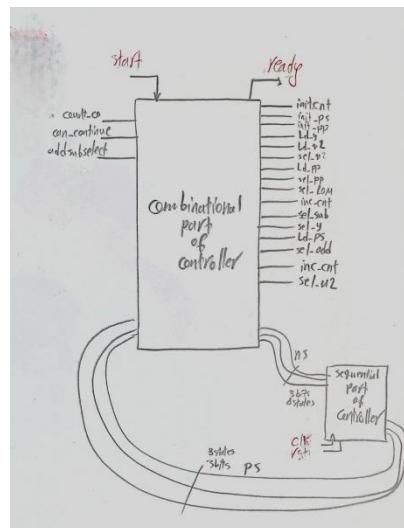
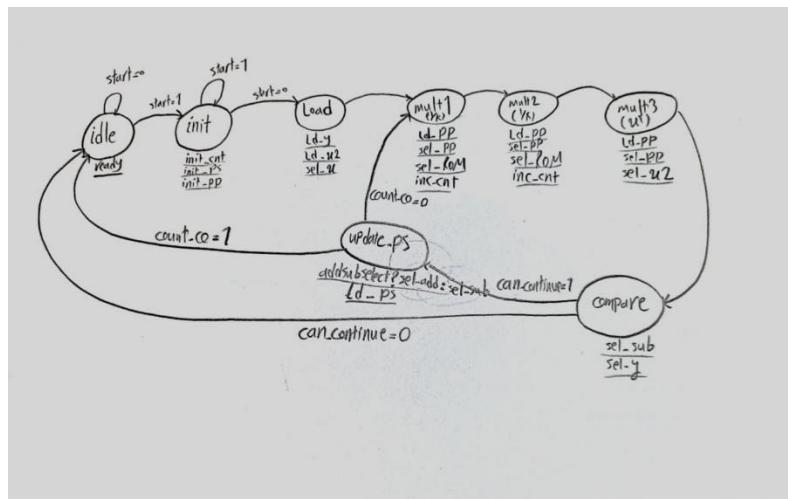
Chip planner:



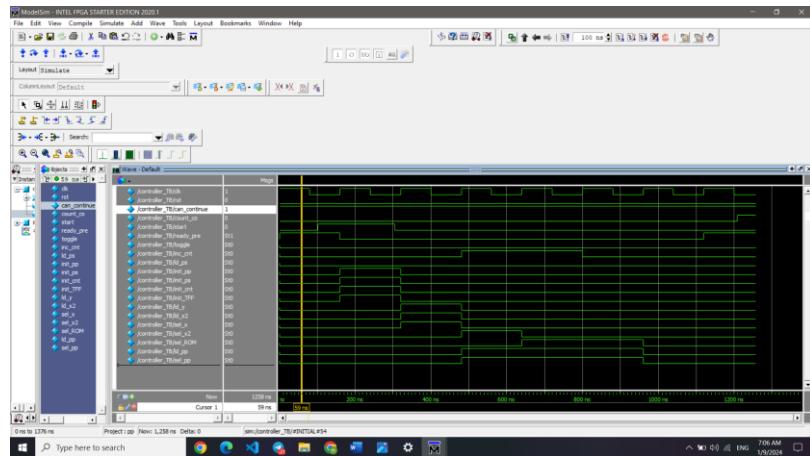
حالا سیمولیت و مقایسه **PRE ,POST**

CONTROLLER

چیز ابتدایی ای که در ذهنم بود این بود:

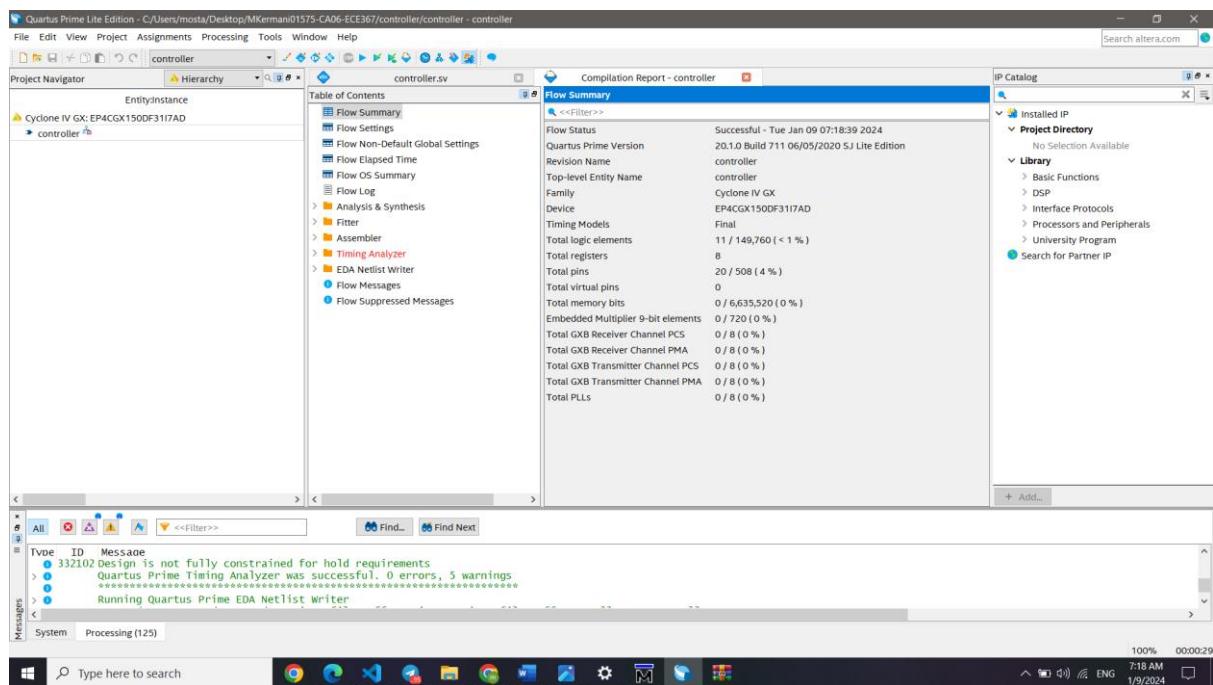


بعد از سیمولیت کردن داریم: به زیبایی بعد از دادن سیگنال شروع میبینیم مراحلی که در کشیده شده بودند، طی میشوند، البته جلوتر با رسم state diagram توسط کوارتز و مقایسه‌ی فایل **.vo** با فایل معمولی بهتر میفهمیم داستان را:

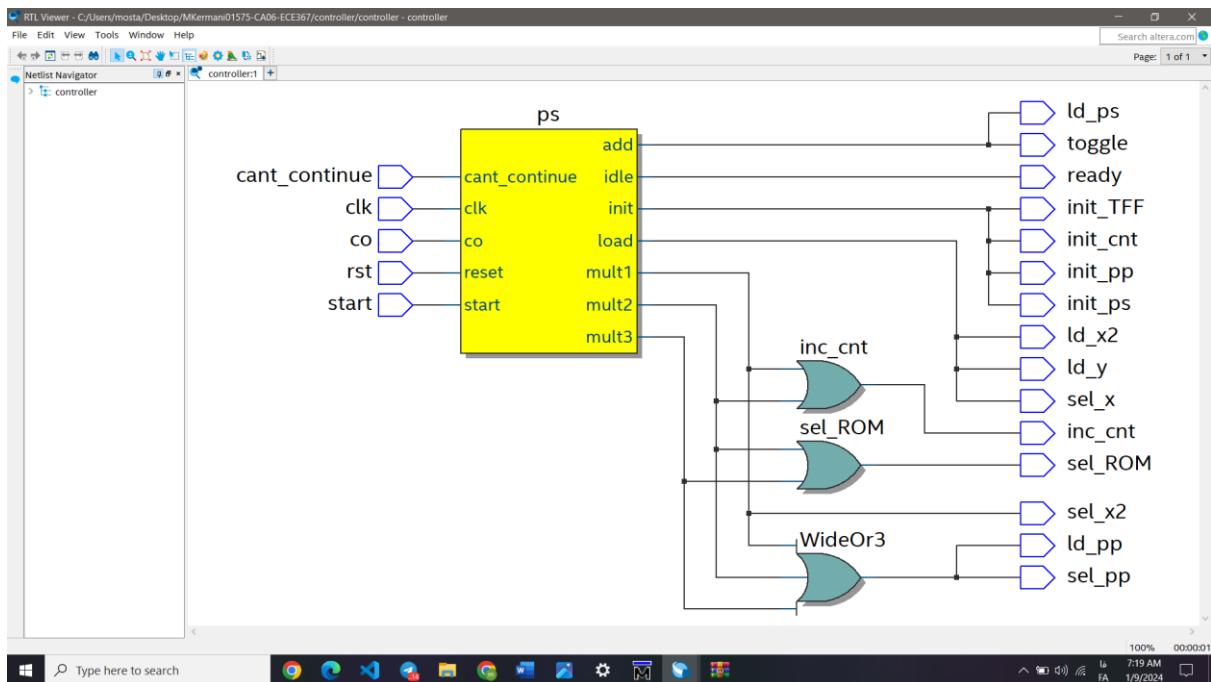


حالا بعد از سیمولیت در کوارتز داریم:

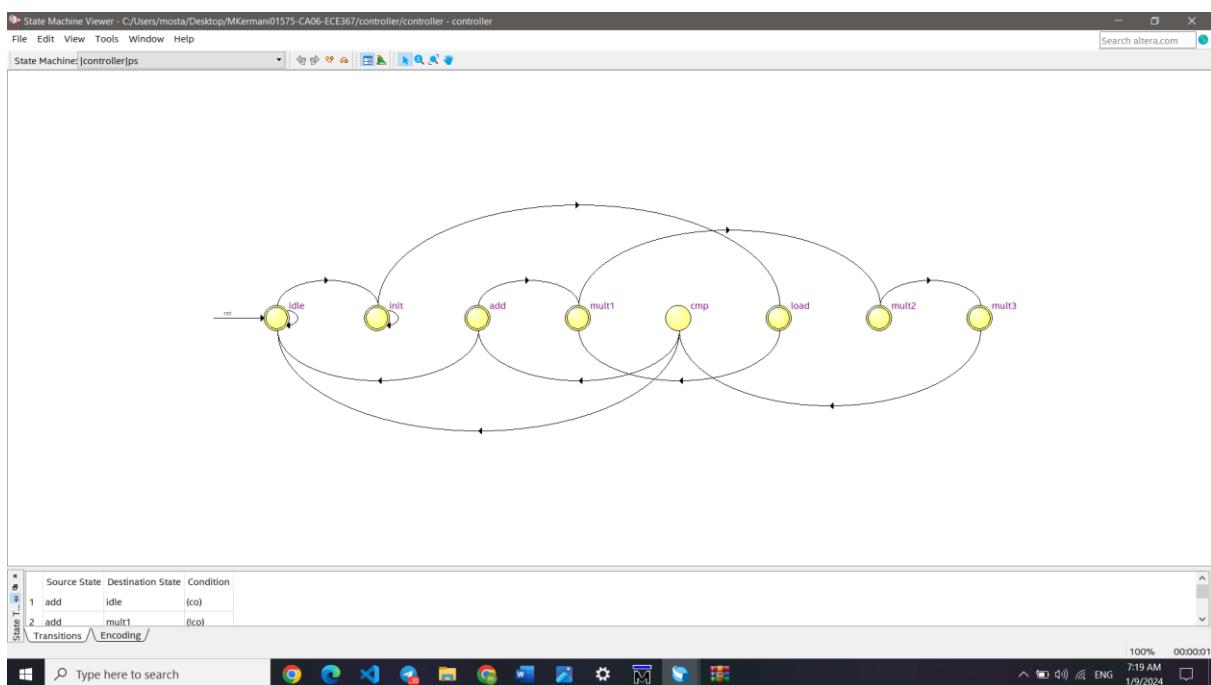
اطلاعات کلی:



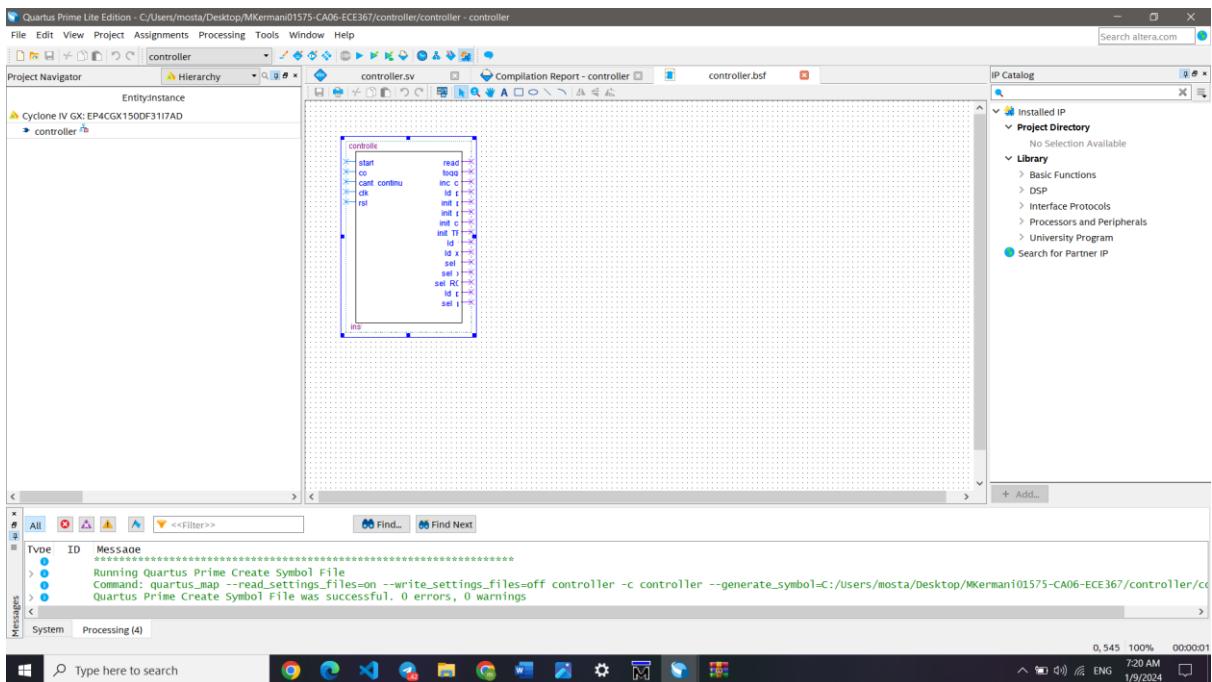
RTL_view:



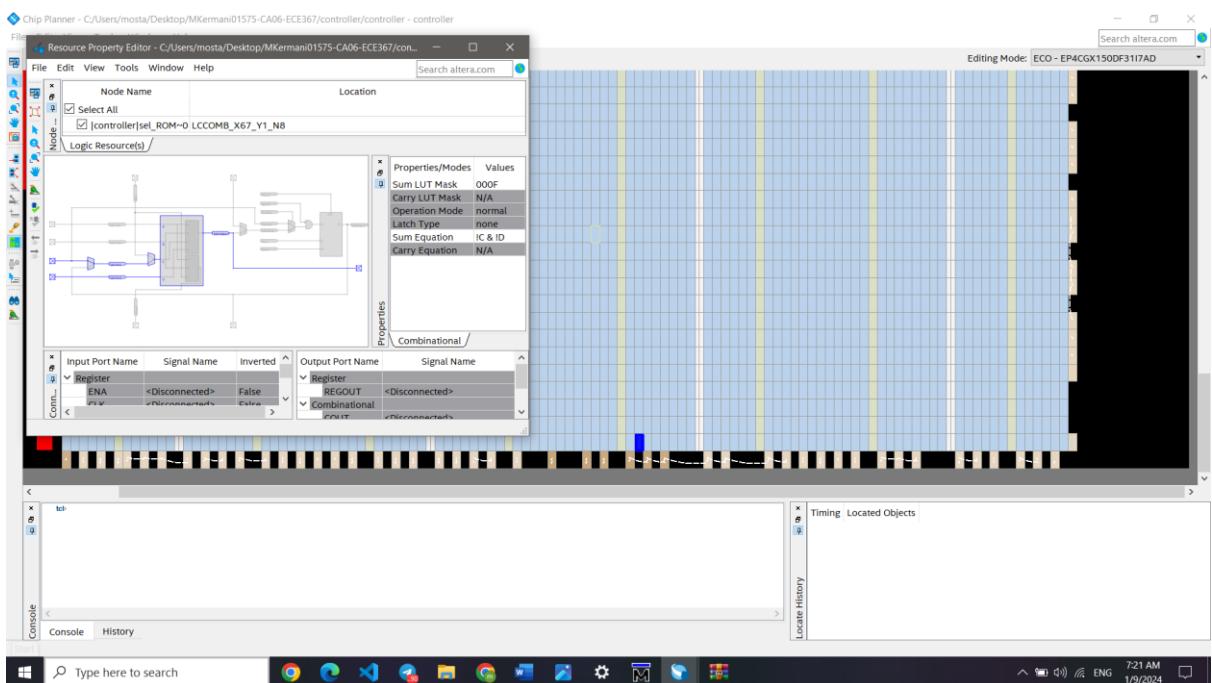
State diagram:



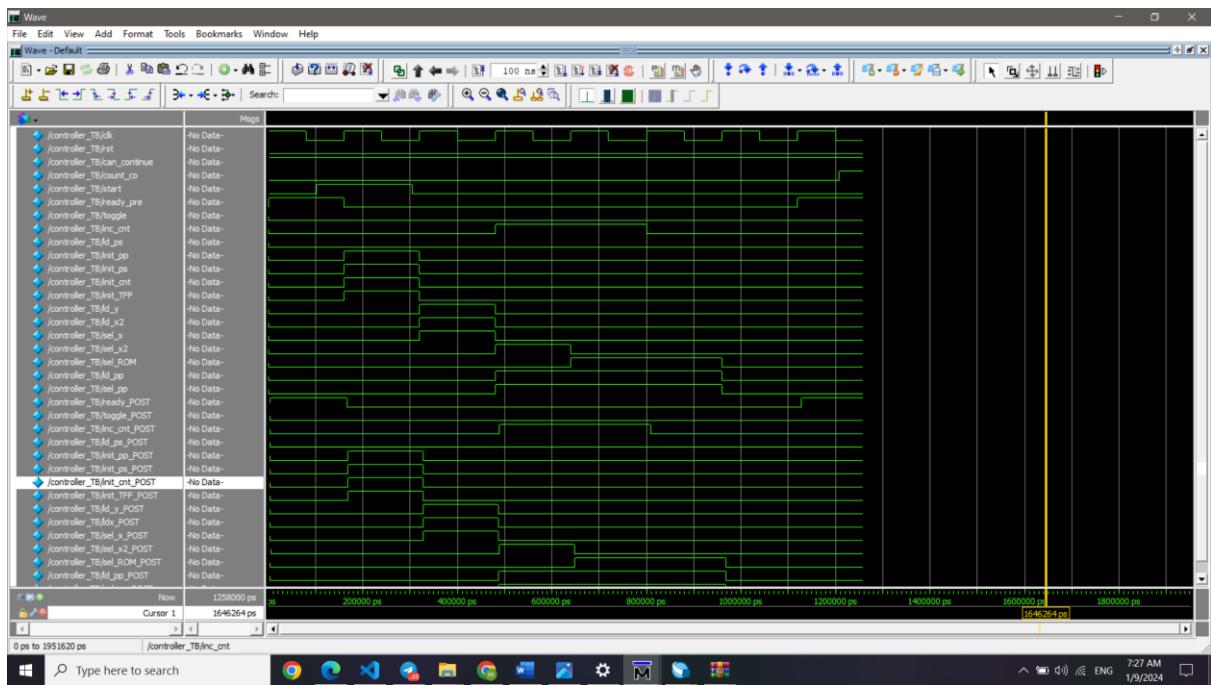
Symbol:



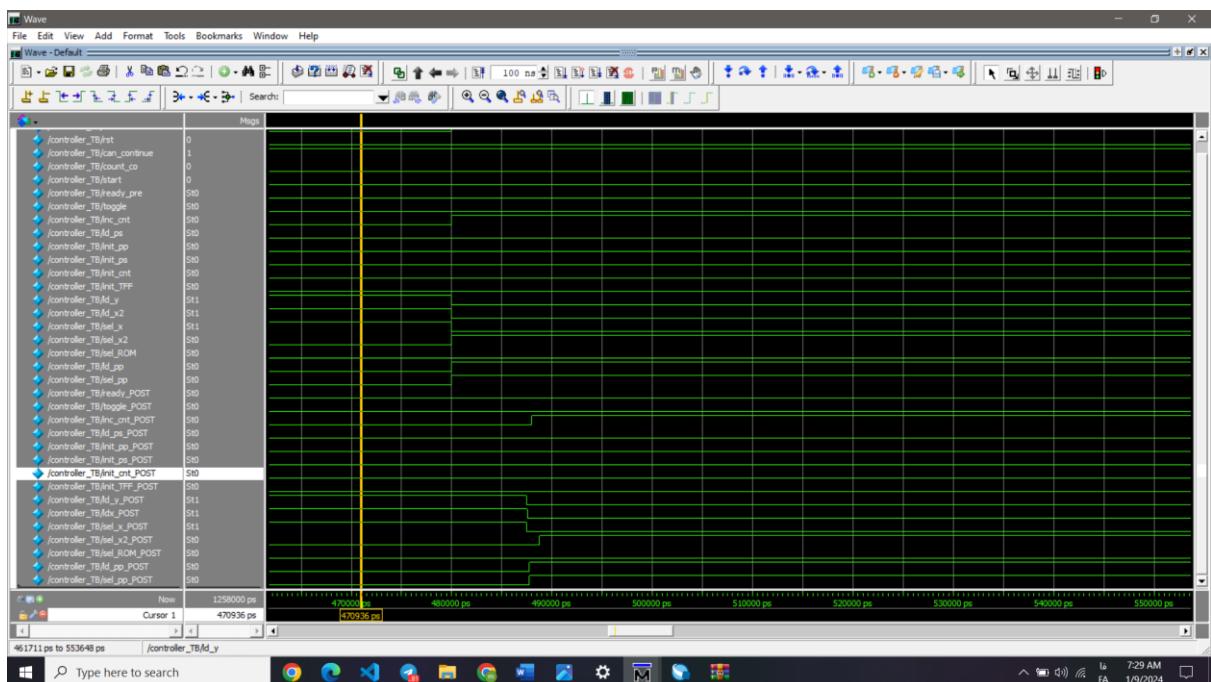
Chip planner:



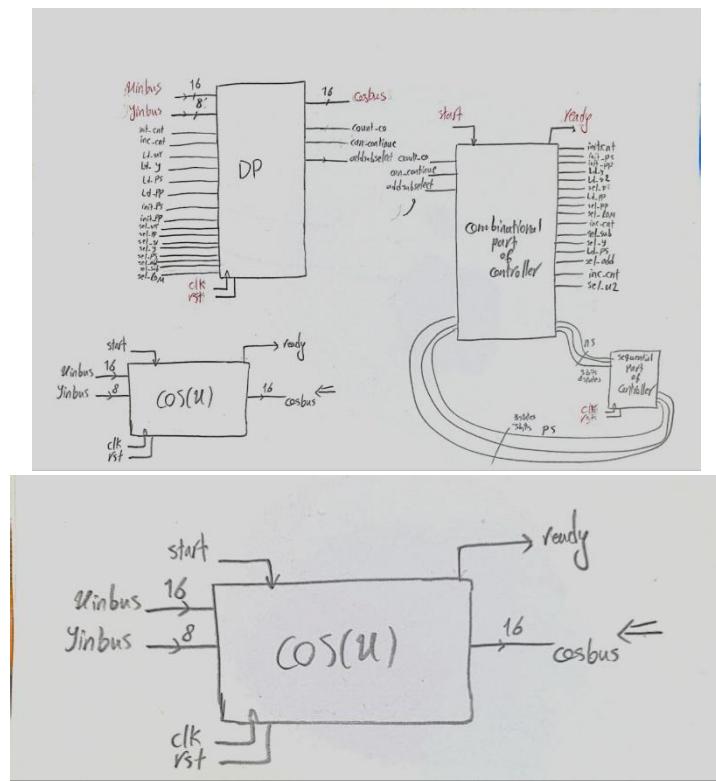
سیمولیت کردن همزمان **pre**, **post** که هر دو مطابقت دارند با **state diagram** یعنی در مرحله اول بعد دیدن استارت، سیگنال های **init** را میزنند و سپس سیگنال های **load** را میزنند و سپس همینطوری مرحله در **state diagram** جلو میروند.



البته با دیلی است :

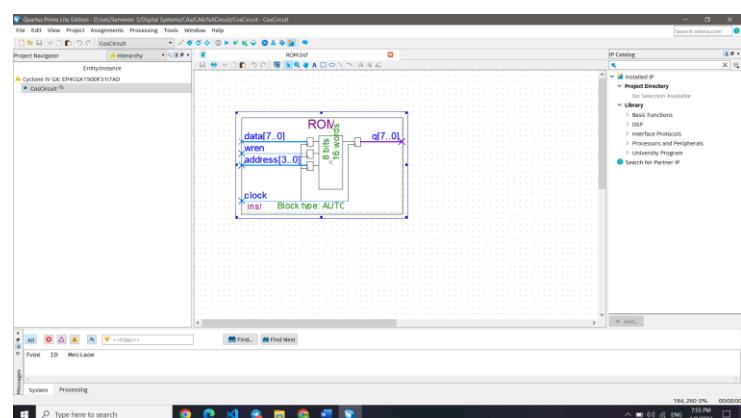


مدار نهایی:



ROM

با لود کردن دستی ورودی



با لود کردن از روی فایل MIF

```
1 WIDTH=8;
2 DEPTH=16;
3 ADDRESS_RADIX=HEX;
4 DATA_RADIX=HEX;
5 CONTENT BEGIN
6 00 : FF; -- memory address : data
7 01 : 80;
8 02 : 55;
9 03 : 40;
10 04 : 33;
11 05 : 2A;
12 06 : 24;
13 07 : 20;
14 08 : 1C;
15 09 : 19;
16 0A : 17;
17 0B : 15;
18 0C : 13;
19 0D : 12;
20 0E : 11;
21 0F : 10;
22 END;
```

کافیست در کوارتز به منوی زیر برویم و رام مدنظر را بسازیم:

روش اول زدن کد وریلاغ است که این شد:

Quartus Prime Lite Edition - C:/Users/mosta/Desktop/present/ROM/ROM - ROM

File Edit View Project Assignments Processing Tools Window Help

ROM

Project Navigator Hierarchy

Table of Contents

Flow Summary

Flow Status: Successful - Wed Jan 10 12:20:13 2024

Quartus Prime Version: 20.1.0 Build 711 06/05/2020 SJ Lite Edition

Revision Name: ROM

Top-level Entity Name: ROM

Family: Cyclone IV GX

Device: EP4CGX150DF3117AD

Timing Models: Final

Total logic elements: 0 / 149,760 (0 %)

Total registers: 0

Total pins: 13 / 508 (3 %)

Total virtual pins: 0

Total memory bits: 32 / 6,635,520 (< 1 %)

Embedded Multiplier 9-bit elements: 0 / 720 (0 %)

Total GXB Receiver Channel PCS: 0 / 8 (0 %)

Total GXB Receiver Channel PMA: 0 / 8 (0 %)

Total GXB Transmitter Channel PCS: 0 / 8 (0 %)

Total GXB Transmitter Channel PMA: 0 / 8 (0 %)

Total PLLs: 0 / 8 (0 %)

Messages

Type ID Message

> 0 Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning

< 293000 Quartus Prime Full Compilation was successful. 0 errors, 19 warnings

System Processing (124)

ROM.v

```

1 module ROM (
2   input [3:0] addr;
3   input clk;
4   output reg [7:0] q;
5 );
6   (* romstyle = "M9K" *) (* ram_init_file = "sin.mif" *) reg [7:0] rom[3:0];
7   always @(posedge clk) begin
8     q <= rom[addr];
9   end
10 endmodule
11
12
13
14
15
16
17

```

Messages

Type ID Message

> 0 Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning

< 293000 Quartus Prime Full Compilation was successful. 0 errors, 19 warnings

System Processing (124)

100% 00:00:30 12:20 PM 1/10/2024

Search altera.com

File Edit View Project Assignments Processing Tools Window Help

ROM

Project Navigator Hierarchy

Table of Contents

Flow Summary

Flow Status: Successful - Wed Jan 10 12:20:13 2024

Quartus Prime Version: 20.1.0 Build 711 06/05/2020 SJ Lite Edition

Revision Name: ROM

Top-level Entity Name: ROM

Family: Cyclone IV GX

Device: EP4CGX150DF3117AD

Timing Models: Final

Total logic elements: 0 / 149,760 (0 %)

Total registers: 0

Total pins: 13 / 508 (3 %)

Total virtual pins: 0

Total memory bits: 32 / 6,635,520 (< 1 %)

Embedded Multiplier 9-bit elements: 0 / 720 (0 %)

Total GXB Receiver Channel PCS: 0 / 8 (0 %)

Total GXB Receiver Channel PMA: 0 / 8 (0 %)

Total GXB Transmitter Channel PCS: 0 / 8 (0 %)

Total GXB Transmitter Channel PMA: 0 / 8 (0 %)

Total PLLs: 0 / 8 (0 %)

Messages

Type ID Message

> 0 Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning

< 293000 Quartus Prime Full Compilation was successful. 0 errors, 19 warnings

System Processing (124)

ROM.v

```

1 module ROM (
2   input [3:0] addr;
3   input clk;
4   output reg [7:0] q;
5 );
6   (* romstyle = "M9K" *) (* ram_init_file = "sin.mif" *) reg [7:0] rom[3:0];
7   always @(posedge clk) begin
8     q <= rom[addr];
9   end
10 endmodule
11
12
13
14
15
16
17

```

Messages

Type ID Message

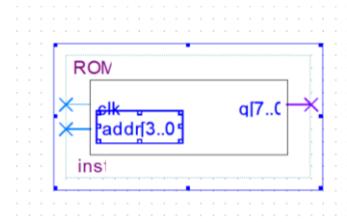
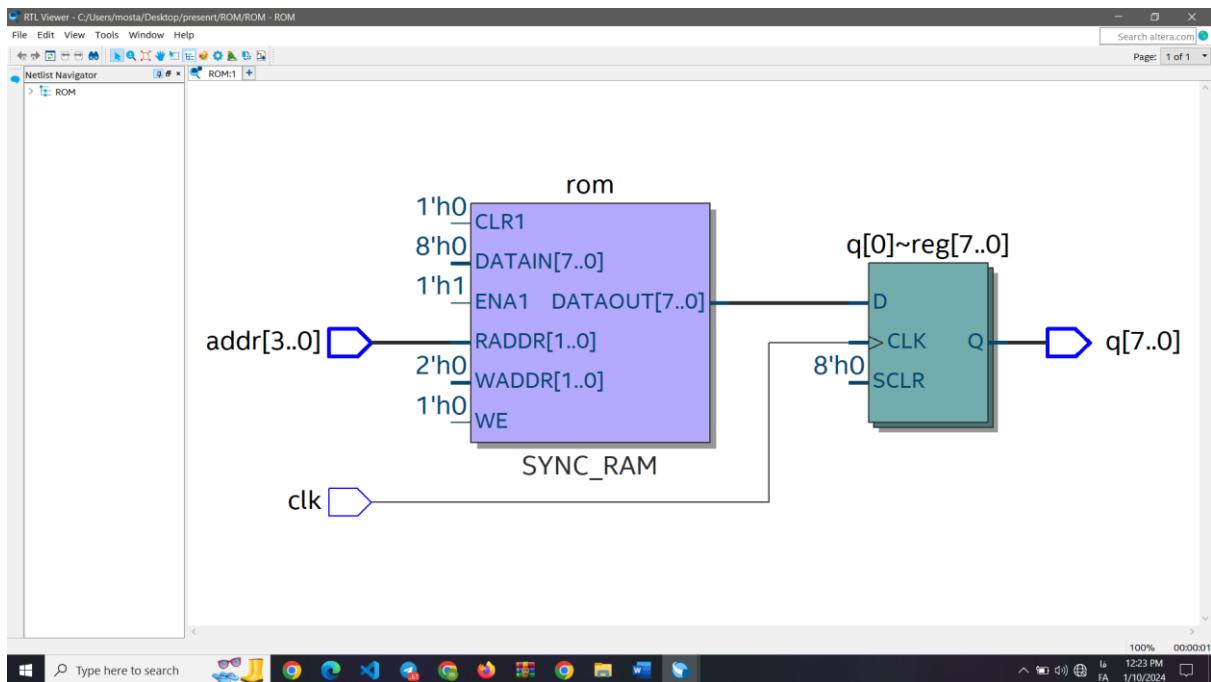
> 0 Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning

< 293000 Quartus Prime Full Compilation was successful. 0 errors, 19 warnings

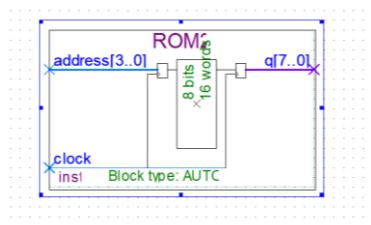
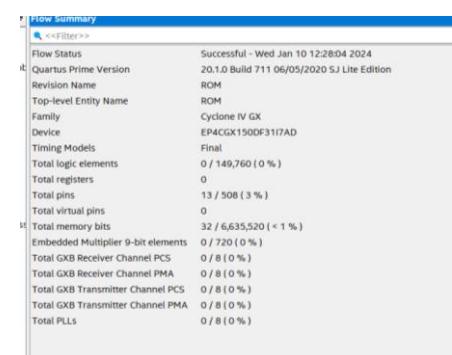
System Processing (124)

100% 00:00:30 12:21 PM 1/10/2024

Search altera.com

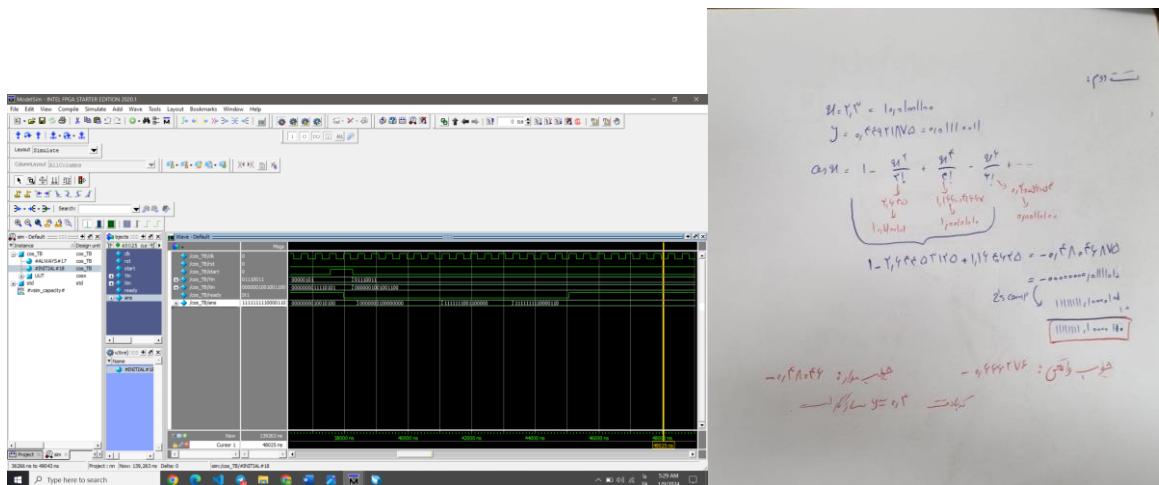
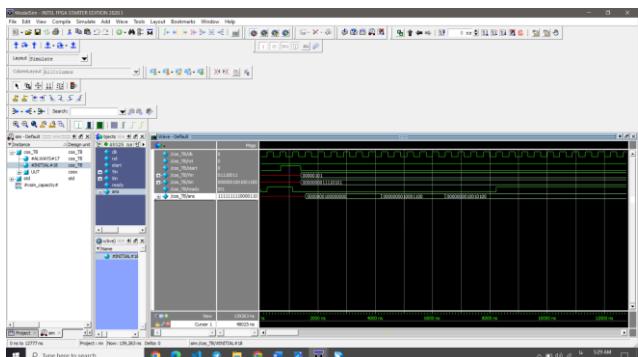


روش دوم با استفاده از این منوی آماده و فقط اپلود کردن mif است:



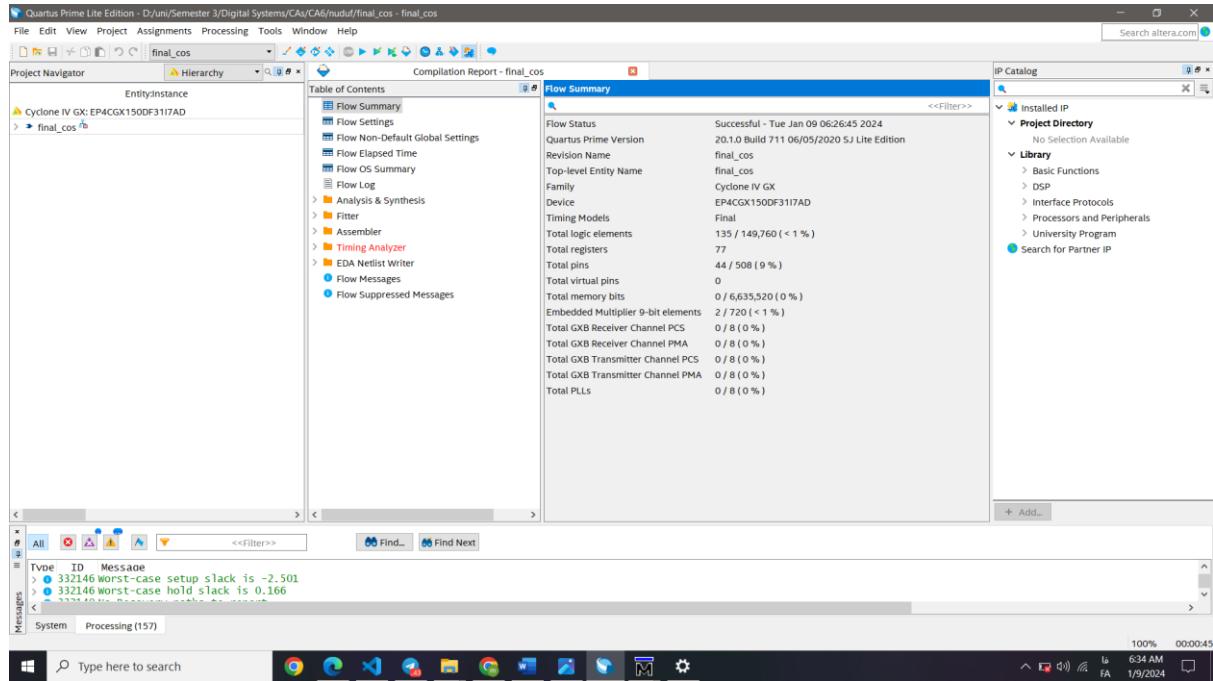
قست کردن مدار نهایی:

$$\begin{aligned} X &= 0.94 = 0.11110101 \\ Y &= 0.19431120 = 0.00001101 \\ \cos 81^\circ &= 1 - \frac{81^2}{2!} + \frac{81^4}{4!} - \frac{81^6}{6!} + \dots \\ &\quad \downarrow \quad \downarrow \quad \downarrow \\ &\quad 0.19431120 \quad 0.00001101 \quad 0.0000001101 \\ &\text{جزء اول است} \rightarrow \text{جزء دوم است} \\ &1 - 0.19431120 + 0.00001101 = 0.1V_0^2 = 0.11110101 \\ &\text{جزء دویم است} \quad \text{جزء چهارم است} \\ &\text{جزء پنجم است} \end{aligned}$$

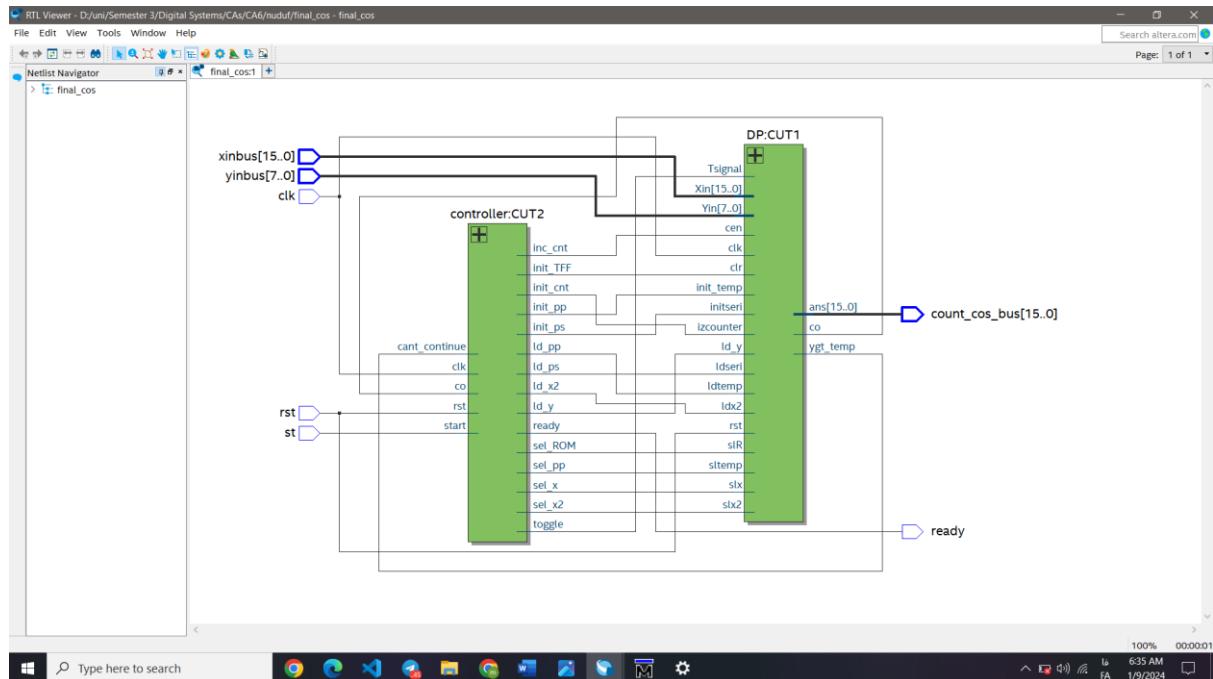


نهایتاً مدار نهایی را در کوارتز سنتز کردم و داریم:

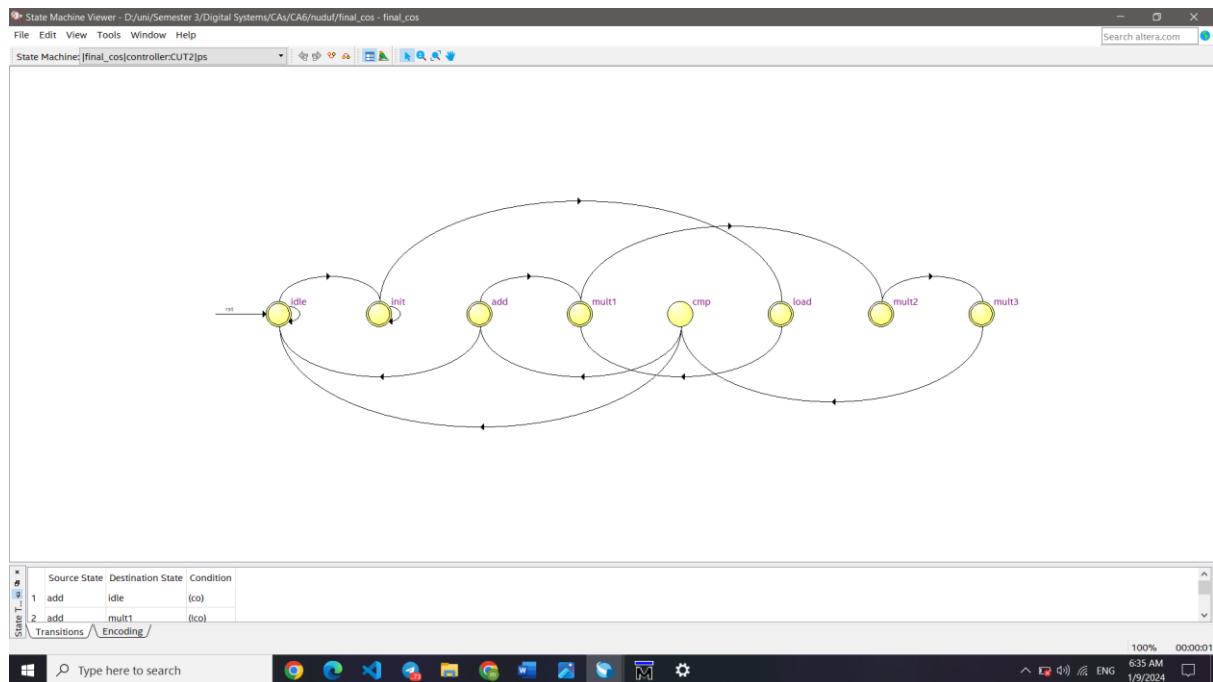
اطلاعات کلی:



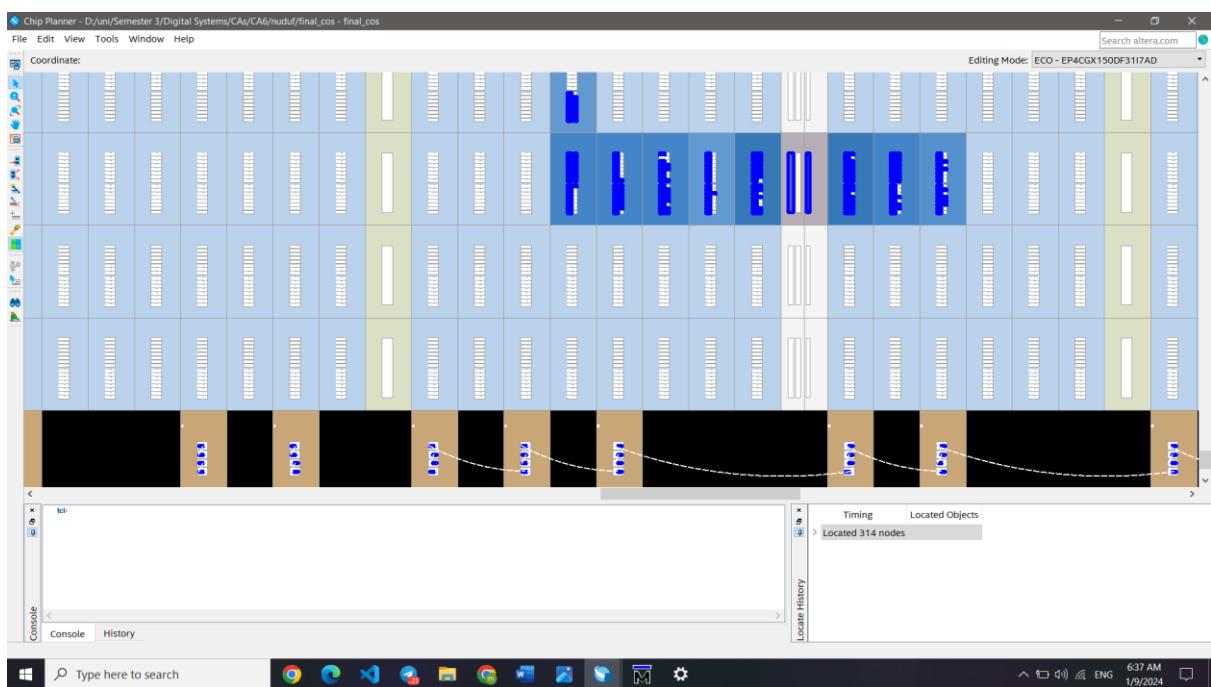
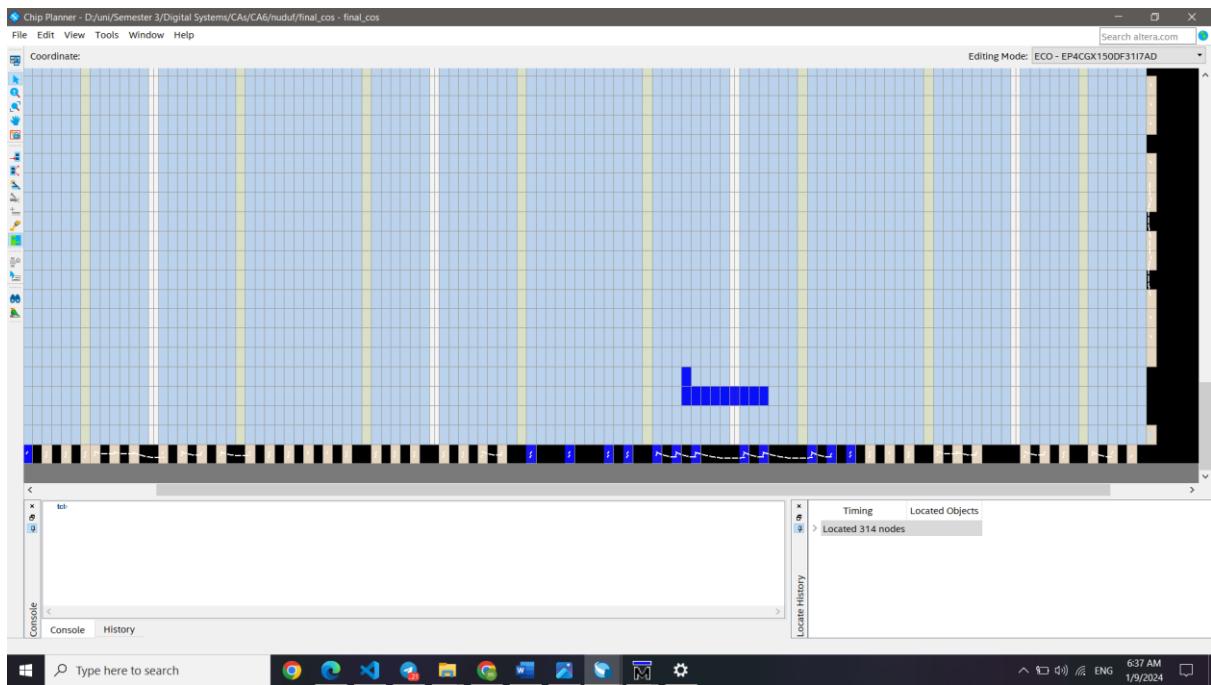
RTL viewer:



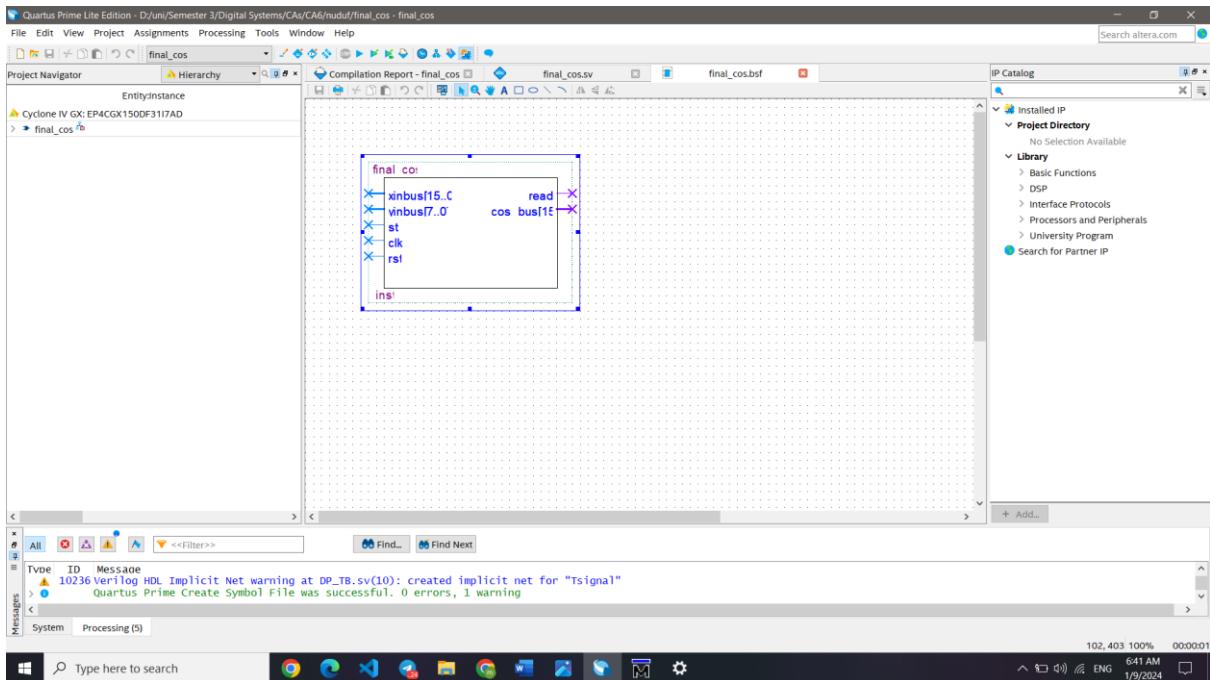
کشیدن مجدد state machine:



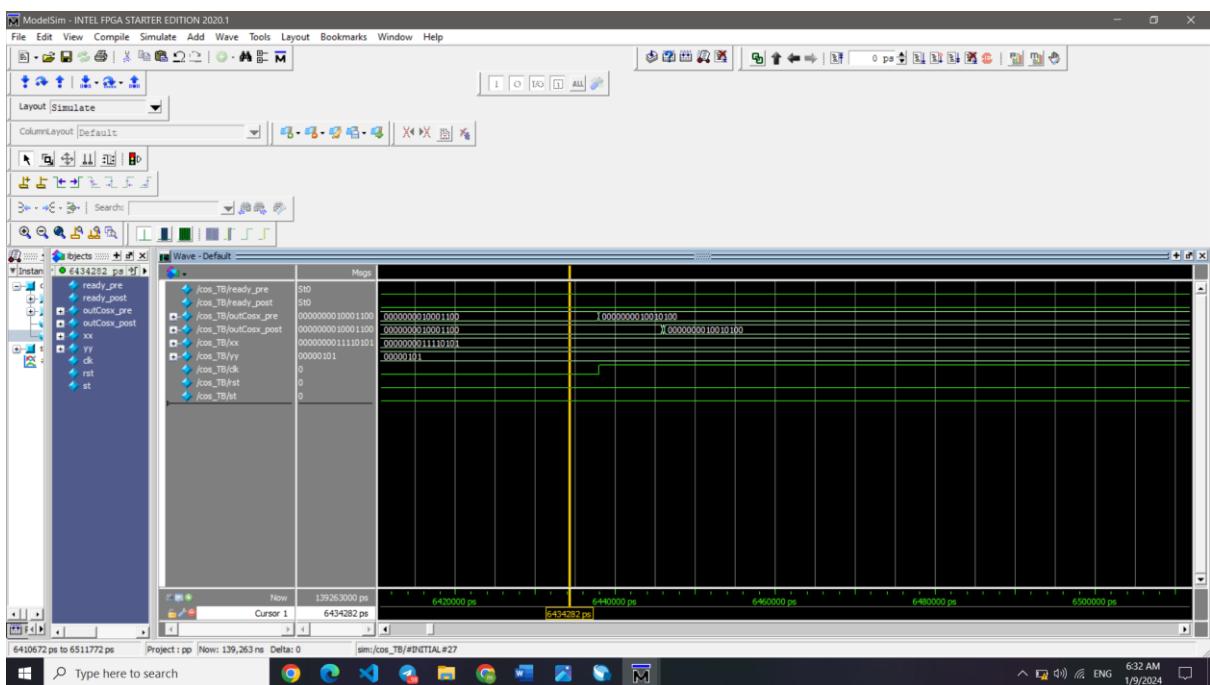
Chip planner:

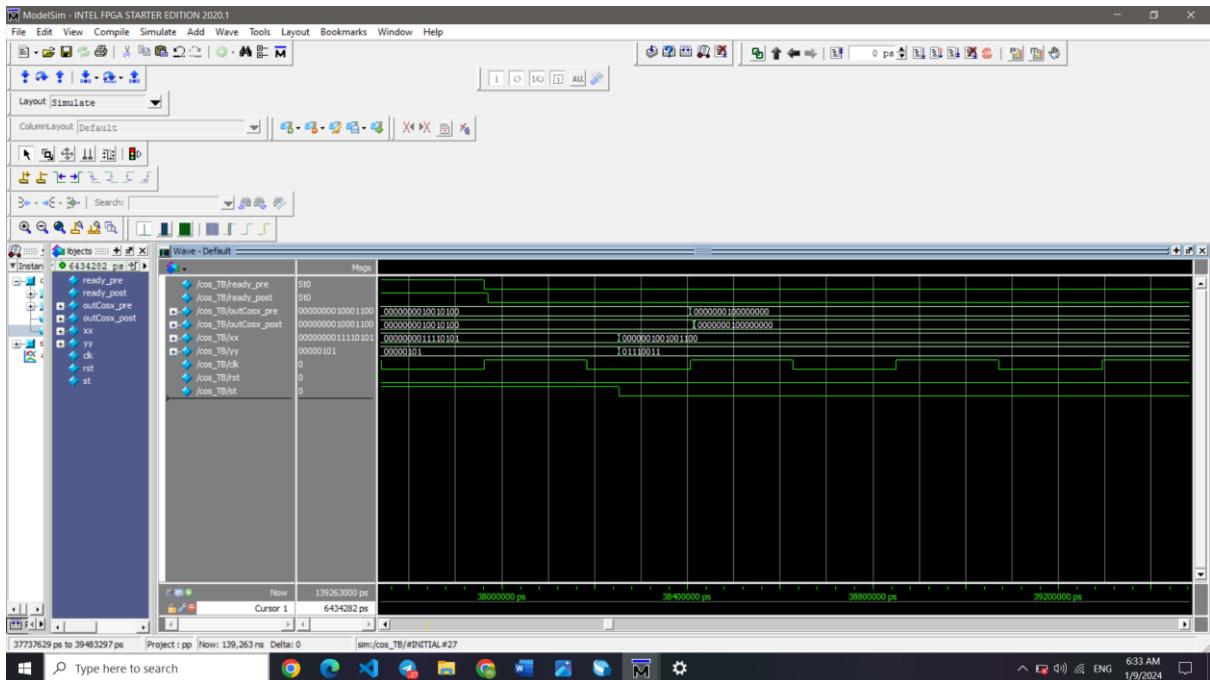


سمبل:



PRE , POST: مقایسه ی





پس همانطور که حدس میزدیم با مقداری تاخیر pre همانند post کار میکند.